CSC 427: Data Structures and Algorithm Analysis

Fall 2006

See online syllabus (also available through Blackboard):

www.dave-reed.com/csc427

Course goals:

- To appreciate the role of algorithms in problem solving and software design; selecting among competing algorithms and justifying choices based on efficiency.
- To understand the specifications and implementations of standard data structures and be able to select appropriate structures in developing programs.
- To develop programs using different problem-solving approaches, and be able to recognize when a particular approach is most useful.
- To be able to design and implement a program to model a real-world system, and subsequently analyze its behavior.
- To recognize the importance of object-oriented techniques, and be able to utilize inheritance and polymorphism to build upon existing code.

1

221 vs. 222 vs. 427

221: programming in the small

- focused on the design & analysis of small programs
- introduced fundamental programming concepts
  - ✓ classes, objects, fields, methods, parameters
  - ✓ variables, assignments, expressions, I/O
  - ✓ control structures (if, if-else, while, for), arrays, ArrayLists

222: programming in the medium

- focused on the design & analysis of more complex programs programs
- introduced more advanced programming & design concepts/techniques
  - ✓ interfaces, inheritance, polymorphism, object composition
  - ✓ searching & sorting, Big-Oh efficiency, recursion, GUIs
  - ✓ stacks, queues, linked lists

you should be familiar with these concepts (we will do some review next week, but you should review your own notes & text)

427: programming in the larger

- focus on complex problems where algorithm/data structure choices matter
- introduce more design techniques, software engineering, performance analysis
  - ✓ object-oriented design, classes & libraries
  - ✓ standard algorithms, big-Oh analysis, problem-solving paradigms
  - ✓ standard collections (lists, stacks, queues, trees, sets, maps)

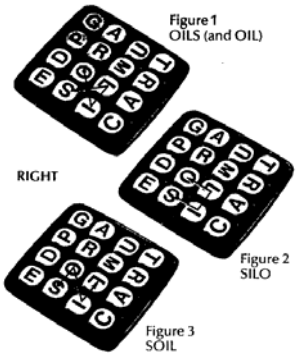
2

When problems start to get complex…

…choosing the right algorithm and data structures are important

- e.g., phone book lookup, checkerboard puzzle

- must develop problem-solving approaches (e.g., divide&conquer, backtracking)
- be able to identify appropriate data structures (e.g., lists, trees, sets, maps)



EXAMPLE: suppose you want to write a program for playing Boggle (Parker Bros.)

- need to be able to represent the board
- need to be able to store and access the dictionary
- need to allow user to enter words
- need to verify user words for scoring
- perhaps show user words they missed

3

Boggle implementations

1. For each user word entered, search the Boggle board for that word.
   - But how do you list all remaining words at the end?

2. Build a list of all dictionary words on the Boggle board by:
   - searching for each word in the dictionary, add to list if on the board.

   For each user word entered, search the list to see if stored (and mark as used).

   At the end, display all words in the list not marked as used.

3. Build a list of all dictionary words on the Boggle board by:
   - exhaustively searching the board, checking letter sequences to see if in the dictionary.

   For each user word entered, search the list to see if stored (and mark as used).

   At the end, display all words in the list not marked as used.

4

Another example…

Sudoku is a popular puzzle craze

given a partially filled in 9x9 grid, place numbers in the grid so that

- each row contains 1..9
- each column contains 1..9
- each 3x3 subsquare contains 1..9

		1			2			
	3	7	8					2
2	4						7	3
4						7	1	
			6		8			
	8	2						9
9	5						3	7
6					5	4	8	
			4			5		

How do people solve these puzzles?

Should a computer program use the same strategies?

- represenation of the grid?
- how fast does the solution need to be?

5

OOP and code reuse

when solving large problems, code reuse is important

- designing, implementing, and testing large software projects is HARD
  whenever possible, want to utilize existing, debugged code
- reusable code is:
  clear and readable (well documented, uses meaningful names, no tricks)
  modular (general, independent routines – test & debug once, then reuse)

OOP is the standard approach to software engineering

philosophy: modularity and reuse apply to data as well as functions

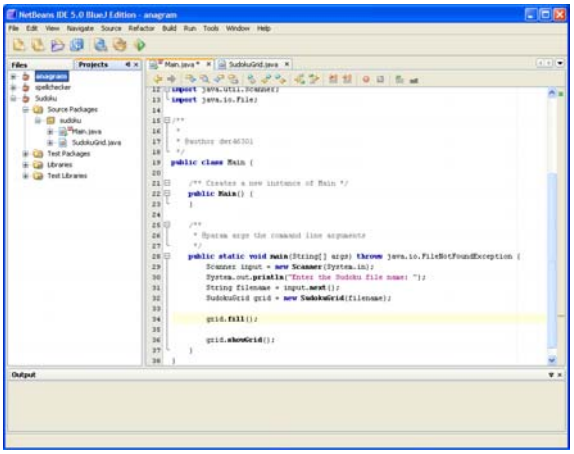
- when solving a problem, must identify the objects involved
  e.g., banking system: customer, checking account, savings account, …
- develop a software model of the objects in the form of abstract data types (ADTs)
  a program is a collection of interacting software objects
  can utilize inheritance to derive new classes from existing ones

6

NetBeans IDE (BlueJ edition)

Sun has released a BlueJ edition of NetBeans

- includes many of the industry-strength features of NetBeans (code completion, refactoring, incremental syntax checking, …)
- simplified interface, designed to ease transition from BlueJ



- for instructions on dowloading & installing a copy (along with the JDK), see:

  cs.creighton.edu/info/software.html

7

Next week…

review of Java

- classes, objects, data fields, methods, parameters
- variables, expressions, assignments, I/O, GUIs, control statements
- strings, arrays, ArrayLists, stacks, queues, linked lists
- searching, sorting, Big-Oh efficiency, recursion

we will go quickly through some examples

- you should review Appendix A and possibly your text/notes from 222

8